#9/Appeal Brief
T. McBeth ...
11/22/03

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Nock, et al.	Docket No.:	RO998-203
Serial No.:	09/259,361	Group Art Unit:	2126
Filed:	02/26/99	Examiner:	Lao, Sue X.

For: APPARATUS AND METHOD FOR COMMUNICATING BETWEEN COMPUTER SYSTEMS USING ACTIVE DATASTREAMS

APPEAL BRIEF

Mail Stop APPEAL BRIEF - PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir/Madam:

This appeal is taken from the Examiner's final rejection, set forth in the Office Action dated 06/18/03, of applicants' claims 1-20. Applicants' Notice of Appeal under 37 C.F.R. § 1.191 was mailed on 09/10/03.

REAL PARTY IN INTEREST

International Business Machines Corporation is the Real Party in Interest.

RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences for this patent application.

STATUS OF CLAIMS

As filed, this case included claims 1-20. In response to the first office action dated 07/17/02, an amendment was filed on 10/17/02 that amended claims 1, 7, 8, 11, 13, 15 and 16. In response to the second office action dated 01/03/03, an amendment was filed on 03/29/03 that amended claims 1, 7, 8, 11, 13, 15 and 16. In the final office action dated 06/18/03, the Examiner rejected claims 1-7 and 16-20 under 35 U.S.C. §103(a) as being unpatentable over WO Patent No. 98/02813 to De Borst *et al.* (hereinafter "De Borst") in view of JDK. Claims 8-15 were rejected under 35 U.S.C. §103(a) as being unpatentable over De Borst in view of JDK and further in view of Sosic. The claims remaining in the case are claims 1-20, all of which stand finally rejected. No claim has been allowed.

STATUS OF AMENDMENTS

The amendments filed on 10/17/02 and 03/29/03 have been entered. Therefore, the claims at issue in this appeal claims 1-20 as amended by the amendment filed on 03/29/03.

SUMMARY OF INVENTION

According to the preferred embodiments of the present invention, two computer systems communicate with each other using active datastreams that each identify executable code for sending and receiving the corresponding active datastream, and for performing any processing required by the active datastream. Each active datastream also includes a unique datastream identifier. When a first computer system (source) has a request to send to a second computer system (target), the source computer system creates an active datastream object that represents the request. Once the active datastream object is created, a method on the active datastream object is invoked to send the active datastream object to the target. A datastream factory on the target reads the datastream identifier of the transmitted active datastream object, determines from the datastream identifier the class corresponding to the type of active datastream object being received, and creates a new instance of the class corresponding to the datastream identifier. A datastream receive mechanism on the target is a method on the new instance, which is invoked to cause the instance to populate itself from the active datastream object received from the source. Logic for replying to the request may also be provided as one or more methods defined on the active datastream class.

ISSUES

The following issues are presented for review on this Appeal:

1. **Whether claims 1-7 and 16-20 are unpatentable as obvious under 35 U.S.C. §103(a) over De Borst in view of JDK**

2. **Whether claims 8-15 are unpatentable as obvious under 35 U.S.C. §103(a) over De Borst in view of JDK and further in view of Sosic**

GROUPING OF CLAIMS

Claims 1, 2, 7, and 16-19 are grouped, and stand and fall together based on claim 1. Claims 3, 10, 14 and 20 are grouped, and stand and fall together based on claim 3. Claims 8, 9, 11 and 12 are grouped, and stand and fall together based on claim 8. It is applicants' intention that the patentability, *vel non*, of claims 4-6, 13 and 15 be considered independently, as these claims do not stand or fall with any other claim.

ARGUMENT

Issue 1: Whether claims 1-7 and 16-20 are unpatentable as obvious under 35 U.S.C. §103(a) over De Borst in view of JDK

Claim 1

In the final office action dated 06/18/03, the Examiner rejected claims 1-7 and 16-20 as obvious under 35 U.S.C. §103(a) over De Borst in view of JDK. Claim 1 recites:

> 1. A computer system comprising:
> at least one processor;
> a memory coupled to the at least one processor;
> a datastream factory residing in the memory and executed by the at least one processor, the datastream factory creating an instance of a datastream class corresponding to an identifier in a datastream received from a second computer system; and
> a datastream receive mechanism residing in the memory and executed by the at least one processor, the datastream receive mechanism defined in the datastream class, the datastream receive mechanism causing the instance of the datastream class to populate itself with information contained in the datastream received from the second computer system by invoking at least one object method on the instance.

The Examiner rejected claim 1 based on the rejection of claim 16. For this reason, the rejection of claim 16 is addressed here. In the rejection of claim 16, the Examiner states that JDK teaches in sections 1.2 and 1.3:

> a newly created datastream object (ObjectInputStream 's') corresponds to an identifier ('in') in a received datastream (InputStream/FileInputStream) such that the received datastream serves as the source datastream for populating (read from the InputStream) (page 3, section 1.3).

Applicants respectfully assert that the teachings of JDK do not read on the creation of an instance of a datastream class *corresponding to an identifier in a received datastream*, as recited in claim 16. Section 1.3 of JDK teaches an input stream called FileInputStream. An ObjectInputStream is then created that provides an interface to the FileInputStream. The Examiner's assertion that the identifier "in" in JDK corresponds to an identifier in a received datastream is incorrect. The pseudo-code that follows the heading 1.3 in JDK shows that the identifier "in" is a variable that is arbitrarily chosen by the programmer to represent the name of an input stream. The ObjectInputStream then uses that variable name as a parameter to its new() constructor method to indicate which data stream the ObjectInputStream provides an interface to. <u>There is no teaching in JDK that the variable name "in" corresponds to an identifier in the received datastream FileInputStream</u>. Any such assertion by the Examiner amounts to sheer assumption and speculation without any support whatsoever in JDK. Applicants respectfully submit that the identifier "in" in the pseudo-code after the heading 1.3 in JDK could be replaced with any suitable variable name, such as GreenGoo. The only purpose of the variable is to name the file input stream in the first line, and to use that name as a parameter in the second line to identify which FileInputStream the ObjectInputStream is an interface to. Thus, if the variable name "in" is replaced with "GreenGoo", the first two lines of pseudo-code would read:

```
FileInputStream GreenGoo = new FileInputStream("tmp");
ObjectInputStream s = new ObjectInputStream(GreenGoo);
```

This shows conclusively that the variable name "in" has no significance in JDK other than providing a name for the FileInputStream that is passed as a parameter in the constructor method for ObjectInputStream. Nowhere does JDK teach or suggest the creation of a datastream class *corresponding to an identifier in a received datastream*, as recited in claim 1. For this reason, claim 1 is allowable over the combination of De Borst and JDK.

In the rejection of claim 16, the Examiner further states that JDK teaches:

> the datastream receive mechanism of JDK (readObject()) . . . causes the newly created datastream object ('s') to populate itself by invoking the object's own method s.readObject() [see section 1.3].

Applicants respectfully assert that the readObject() method in JDK does not cause the datastream object s to populate itself by invoking the s.readObject(). We now look at the express teachings of JDK to determine what it does and does not teach.

JDK states at the first paragraph following the pseudo-code that follows the 1.3 heading on page 3:

> First an InputStream, in this case a FileInputStream, is needed as the source stream. Then an ObjectInputStream is created that reads from the InputStream. Next, the string "Today" and a Date object are read from the stream. Generally, objects are read with the readObject method and primitives are read from the stream with the method of DataInput.

JDK thus teaches a stream FileInputStream that contains streamed data and an ObjectInputStream that provides an interface for accessing data within FileInputStream. The readObject() method on the ObjectInputStream class is a method that allows extracting data from the stream. This is shown clearly from the pseudo-code in JDK:

```
String today = (String)s.readObject();
Date date = (Date)s.readObject();
```

In the first line above, a string labeled "today" is read from the ObjectInputStream using the readObject() method. In the second line above, a date "date" is read from the ObjectInputStream using the readObject() method. In both cases, data is being read from the stream by invoking methods on the ObjectInputStream. The readObject() method is thus provided as a tool to extract data from the input stream. The readObject() method is not used in JDK to populate an instance of a datastream class. To the contrary, the readObject() method in JDK is used to extract data from an instance of a datastream class. In JDK, the ObjectInputStream simply provides an interface for accessing the data in the FileInputStream. This means the ObjectInputStream is not populated with data from the FileInputStream, but simply provide an interface (*i.e.*, the readObject() method) which accesses data in the FileInputStream.

JDK clearly shows the state of the art in datastreams before applicants' invention, as illustrated in FIG. 1 of applicants' disclosure. In order to extract data from the stream, the code receiving the stream must have a hard-coded datastream specification that specifies the format and order of objects within the stream. In the simple example provided in JDK, namely:

```
String today = (String)s.readObject();
Date date = (Date)s.readObject();
```

the receiving computer system uses its hard-coded datastream specification to determine the order of objects in the input stream. If these two instructions were in reverse order with the same datastream, the first s.readObject() method call would expect a date, but the object returned would be a string, leading to an error. The second s.readObject() method call would expect a string, but the object returned would be a date, leading to an error. One can see from this simple example in JDK that the intelligence regarding what's in the stream must reside in both the sending computer system and in the receiving computer system. In JDK, if a new datastream is defined, the new datastream definition must be added to the hard-coded datastream specification on both the sending computer system and the receiving computer system before the datastream may be sent. In the claimed invention, in contrast, the intelligence regarding what's in a stream can be encapsulated in the stream itself, because the instance of the datastream class is populated from the input stream by invoking a method on the datastream class. As a result, the target computer system need not contain any intelligence regarding the specific configuration or format for the incoming data stream, because the code to create a datastream instance from the stream resides in the stream itself. This is a significant difference between the claimed invention and JDK. Nowhere does JDK teach or suggest a "datastream receive mechanism defined in the datastream class, the datastream receive mechanism causing the instance of the datastream class to *populate itself* with information contained in the datastream received from the second computer system by invoking at least one object method on the instance." Because JDK does not teach the datastream receive mechanism recited in claim 1, claim 1 is allowable over the combination of De Borst and JDK. Applicants respectfully request that the Examiner's rejection of claim 1 under 35 U.S.C. §103(a) be reversed.

Claims 2, 7 and 16-19

Claims 2, 7 and 16-19 are grouped with claim 1, and stand or fall with claim 1.

Claim 3

The arguments above with respect to claim 1 apply equally to claim 3, and are incorporated in this section by reference. Claim 3 recites:

> 3. The computer system of claim 1 further comprising a datastream send mechanism residing in the memory for sending the instance of the datastream by invoking at least one object method on the instance.

In rejecting claim 3, the Examiner relies upon the rejection of claim 20. For this reason, the rejection of claim 20 is considered here. In rejecting claim 20, the Examiner cites to the Put operation of De Borst as allegedly teaching the datastream send mechanism in claim 3. Note, however, that the datastream send mechanism in claim 3 sends the instance of the datastream by invoking at least one object method on the instance. In other words, the datastream instance has a method that causes the datastream instance to send itself. The Put operation in De Borst allows *writing* to a datastream, but does not *send* the instance of a datastream class. The mechanisms to send a datastream are discussed in De Borst at p. 19 line 11 to p. 20 line 16. The transporter component in De Borst is separate from the datastream component. De Borst thus expressly teaches away from the limitations in claim 3 that allow a datastream send mechanism to send the instance of the datastream by invoking at least one object method on the instance. The Put operation in De Borst causes one piece of information to be written to a datastream. It does not cause the datastream to be sent, and it does not cause an instance of a datastream class to be sent. For these many reasons, claim 3 is allowable over the combination of De Borst and JDK. In addition, claim 3 depends on claim 1, which is allowable for the reasons given above. As a result, claim 3 is also allowable as depending on an allowable independent claim. Applicants respectfully request that the Examiner's rejection of claim 3 under 35 U.S.C. §103(a) be reversed.

Claim 20

Claim 20 is grouped with claim 3, and stands and falls with claim 3.

Claim 4

The arguments above with respect to claim 1 apply equally to claim 4, and are incorporated in this section by reference. Claim 4 recites:

> 4. The computer system of claim 1 wherein the datastream identifies executable code residing in the memory for receiving the datastream from the second computer system.

In rejecting claim 4, the Examiner simply states: "As to claim 4, it is covered by claim 1 (datastream receive mechanism ... executed by)." This rejection makes no sense. If the limitations in claim 4 are "covered" by claim 1 upon which claim 4 depends, the Examiner is essentially stating that claim 4 does not further limit claim 1. This is nonsense. Claim 4 recites that the datastream identifies executable code residing in the memory for receiving the datastream from the second computer system. The Examiner has failed to address these limitations in claim 4, and has therefore failed to establish a prima facie case of obviousness for claim 4 under 35 U.S.C. §103(a). Nowhere does De Borst nor JDK teach or suggest that the datastream itself identifies *executable code for receiving the datastream from the second computer system*. For this reason, claim 4 is allowable over the combination of De Borst and JDK. In addition, claim 4 depends on claim 1, which is allowable for the reasons given above. As a result, claim 4 is also allowable as depending on an allowable independent claim. Applicants respectfully request that the Examiner's rejection of claim 4 under 35 U.S.C. §103(a) be reversed.

<u>Claim 5</u>

The arguments above with respect to claims 1 and 4 apply equally to claim 5, and are incorporated in this section by reference. Claim 5 recites:

> 5. The computer system of claim 4 wherein the datastream further identifies executable code residing in the memory for performing a request represented by the datastream.

In rejecting claim 5, the Examiner cites to his rejection of claim 19, which is addressed here. In rejecting claim 19, the Examiner states: "As to claim 19, Borst teaches datastream processing mechanism (Get, Put, Destroy and Cancel operations, page 12, lines 22-23)." While it is true that the operations cited by the Examiner are executable code, the Examiner has failed to address the rest of the limitations in claim 5. The executable code in claim 5 is for performing a request *represented by the datastream.*

Because the Examiner has not addressed this limitation, the Examiner has failed to establish a prima facie case of obviousness for claim 5 under 35 U.S.C. §103(a).

Nowhere does De Borst nor JDK teach or suggest that a datastream may represent a request. Furthermore, neither teach or suggest executable code identified by the datastream for performing the request. For these reasons, claim 5 is allowable over the combination of De Borst and JDK. In addition, claim 5 depends on claim 4, which depends on claim 1, which is allowable for the reasons given above. As a result, claim 5 is also allowable as depending on an allowable independent claim. Applicants respectfully request that the Examiner's rejection of claim 5 under 35 U.S.C. §103(a) be reversed.

Claim 6

The arguments above with respect to claims 1, 4 and 5 apply equally to claim 6, and are incorporated in this section by reference. Claim 6 recites:

> 6. The computer system of claim 5 wherein the datastream further identifies executable code residing in the memory for sending the datastream from the second computer system to the computer system.

In rejecting claim 6, the Examiner cites to the rejection of claim 3, which cites to the rejection of claim 20. In rejecting claim 20, the Examiner cites to the Put operation of De Borst as allegedly teaching the limitations in claim 6. Note, however, that the limitations in claim 6 include executable code that is identified by the datastream for sending the datastream from the second computer system to the computer system. None of these limitations have been addressed by the Examiner in his rejection of claim 6. As a result, the Examiner has failed to establish a prima facie case of obviousness for claim 6 under 35 U.S.C. §103(a).

Note that claim 6 depends on claim 5, which depends on claim 4. Claim 4 recites that the datastream identifies executable code for ***receiving the datastream*** from the second computer system. Claim 6, in contrast, recites that the datastream additionally identifies executable code for ***sending the datastream*** from the second computer system to the computer system. Thus we see in claim 6 executable code for sending the datastream, while in claim 4 we see executable code for receiving the datastream. Neither De Borst nor JDK teach or suggest a datastream that identifies both executable code for sending the datastream and executable code for receiving the datastream. For this reason, claim 6 is allowable over the combination of De Borst and JDK. In addition, claim 6 depends on claim 5, which depends on claim 4, which depends on claim 1, which is allowable for the reasons given above. As a result, claim 6 is also allowable as depending on an allowable independent claim. Applicants respectfully request that the Examiner's rejection of claim 6 under 35 U.S.C. §103(a) be reversed.

Issue 2: Whether claims 8-15 are unpatentable as obvious under 35 U.S.C. §103(a) over De Borst in view of JDK and further in view of Sosic

In rejecting claims 8-15, the Examiner states:

> Sosic teaches stream operations, wherein a datastream is an active datastream (executable stream) which identifies executable code for processing (program counter evpc, event instruction evinst) the active datastream (process even in the executions stream).

Applicants strongly assert that Sosic has nothing whatsoever to do with datastreams. The "execution stream" in Sosic provides the description of the executor's computational behavior. The fact that it is called an "execution stream" does not mean that Sosic teaches a datastream within the scope of that term as used in the claims. In fact, a sequence of instructions that are executed by a processor are often referred to in the art as an "instruction stream", meaning a stream (or series) of instructions that are executed. But clearly, this type of instruction stream has no relevance whatsoever to the datastream in the claims. The execution stream in Sosic is part of a programming environment for developing computer programs. One of ordinary skill in the art would not be motivated to combine the execution stream in Sosic that relates to a programming environment to the field of datastreams taught in De Borst and JDK. For this reason, the combination of De Borst, JDK and Sosic is improper.

Claim 8

The arguments above with respect to claim 1 applies equally to claim 8, and are incorporated in this section by reference. Nowhere does any of De Borst, JDK, Sosic, nor their combination teach or suggest "means for constructing an active datastream, *the active datastream including a datastream identifier that identifies executable code for processing the active datastream*" as recited in claim 8. For Sosic to read on this limitation, the execution stream in Sosic would have to include a datastream identifier that identifies executable code for processing the execution stream. Sosic has no such teaching or suggestion.

In addition, none of the cited references teach or suggest means defined in the datastream class for causing the instance of the datastream class to populate itself with information contained in the active datastream received from the first computer system. For these reasons, claim 8 is allowable over the combination of De Borst, JDK and Sosic. Applicants respectfully request that the Examiner's rejection of claim 8 under 35 U.S.C. §103(a) be reversed.

Claims 9, 11 and 12

Claims 9, 11 and 12 are grouped with claim 8, and stand and fall with claim 8.

Claims 10 and 14

Claims 10 and 14 are grouped with claim 3, and stand and fall with claim 3.

Claim 13

The arguments above with respect to claims 1 and 8 apply equally to claim 13, and are incorporated in this section by reference. Claim 13 recites:

> 13. The method of claim 11 wherein the step of the instance of the datastream class populating itself with the information contained in the active datastream includes the step of executing a receive method on the instance of the datastream class.

In rejecting claim 13, the Examiner cites to the receive mechanism in the rejection of claim 16 as allegedly reading on claim 13. The readObject() method cited by the

Examiner in JDK causes one object to be read from the datastream object. The readObject() method in JDK does not cause an instance of the datastream class to *populate itself* with information in the datastream. For this reason, JDK does not teach the limitations in claim 13, so claim 13 is allowable over the combination of De Borst, JDK and Sosic. In addition, claim 13 depends on claim 11, which is allowable for the reasons given above. As a result, claim 13 is also allowable as depending on an allowable independent claim. Applicants respectfully request that the Examiner's rejection of claim 13 under 35 U.S.C. §103(a) be reversed.

Claim 15

The arguments above with respect to claims 1 and 8 apply equally to claim 15, and are incorporated in this section by reference. Claim 15 recites:

> 15. A method for communicating between a first computer system and a second computer system, the method comprising the steps of:
> the first computer system constructing an active datastream object, the active datastream object including a datastream identifier that identifies a corresponding datastream class that includes executable code corresponding to a plurality of object methods for processing the active datastream object;
> the first computer system sending the active datastream to the second computer system by invoking a send method on the active datastream object;
> the second computer system reading the datastream identifier from the active datastream object received from the first computer system;
> the second computer system creating a new instance of the datastream class that corresponds to the datastream identifier;
> the new instance of the datastream class populating itself with information contained in the active datastream received from the first computer system by invoking a receive method on the new instance; and
> the second computer system performing a request represented by the active datastream by invoking at least one object method on the new instance.

In rejecting claim 15, the Examiner states:

> As to claim 15, note the rejections of claims 11-14. It is noted that an active datastream object identifying its datastream class would have been obvious based on object naming principle in object-oriented programming.

Applicants assert that the cursory rejection of claim 15 above does not address all of the limitations in claim 15. In particular, the following limitations in claim 15 have not been addressed by the Examiner: the plurality of object methods at lines 5 and 6; the send method at line 8; all limitations at lines 9 and 10; and all limitations at lines 16 and 17. Because the Examiner has failed to address these limitations, which are not contained in any of claims 11-14, the Examiner has failed to establish a prima facie case of obviousness for claim 15 under 35 U.S.C. §103(a).

Claim 15 specifically recites at lines 9-10 "the second computer system reading the datastream identifier from the active datastream object received from the first computer system". This limitation has not been addressed by the Examiner. Note that the identifier corresponds to a datastream class (see lines 4-5), and is used to create a new instance of the datastream class that corresponds to the identifier at lines 11-12. Nowhere does De Borst, JDK nor Sosic teach or suggest sending a datastream identifier that identifies a corresponding datastream class as part of the datastream, reading the identifier from the datastream object, and creating a new instance of the datastream class that corresponds to the datastream identifier.

Applicants respectfully assert that the Examiner has not considered claim 15 as a whole. He has found bits and pieces from three different references, but these bits and pieces do not read on all of the limitations in the claims. When claim 15 is considered as a whole, the combination of De Borst, JDK and Sosic do not render obvious the specific steps recited in claim 15. Applicants respectfully request that the Examiner's rejection of claim 15 under 35 U.S.C. §103(a) be reversed.

CONCLUSION

Claims 1-20 are addressed in this Appeal. For the numerous reasons articulated above, applicants maintain that the rejection of claims 1-20 under 35 U.S.C. § 103(a) is erroneous.

Applicants respectfully submit that this Appeal Brief fully responds to, and successfully contravenes, every ground of rejection and respectfully requests that the final rejection be reversed and that all claims in the subject patent application be found allowable.

Respectfully submitted,

[illegible]

Derek P. Martin
Reg. No. 36,595

MARTIN & ASSOCIATES, L.L.C.
P.O. Box 548
Carthage, MO 64836-0548
(417) 358-4700
Fax (417) 358-5757

APPENDIX - CLAIMS FINALLY REJECTED

1. A computer system comprising:

at least one processor;

a memory coupled to the at least one processor;

a datastream factory residing in the memory and executed by the at least one processor, the datastream factory creating an instance of a datastream class corresponding to an identifier in a datastream received from a second computer system; and

a datastream receive mechanism residing in the memory and executed by the at least one processor, the datastream receive mechanism defined in the datastream class, the datastream receive mechanism causing the instance of the datastream class to populate itself with information contained in the datastream received from the second computer system by invoking at least one object method on the instance.

2. The computer system of claim 1 further comprising a datastream processing mechanism residing in the memory for processing the instance of the datastream by invoking at least one object method on the instance.

3. The computer system of claim 1 further comprising a datastream send mechanism residing in the memory for sending the instance of the datastream by invoking at least one object method on the instance.

4. The computer system of claim 1 wherein the datastream identifies executable code residing in the memory for receiving the datastream from the second computer system.

5. The computer system of claim 4 wherein the datastream further identifies executable code residing in the memory for performing a request represented by the datastream.

6. The computer system of claim 5 wherein the datastream further identifies executable code residing in the memory for sending the datastream from the second computer system to the computer system.

7. A networked computer system comprising:

a first computer system coupled via a network connection to a second computer system;

each of the first and second computer systems comprising a datastream processor, the datastream processor including:

a datastream factory for creating an instance of an active datastream class corresponding to a datastream identifier received in a datastream on the network connection from the other computer system; and

a datastream receive mechanism defined in the active datastream class that causes the instance of the active datastream class to populate itself with information contained in the datastream received on the network connection from the other computer system by invoking at least one object method on the instance.

8. A networked computer system comprising:

a first computer system coupled via a network connection to a second computer system;

means for constructing an active datastream, the active datastream including a datastream identifier that identifies executable code for processing the active datastream;

means for sending the active datastream from the first computer system to the second computer system;

means for creating an instance of a datastream class that corresponds to the datastream identifier in the second computer system;

means defined in the datastream class for causing the instance of the datastream class to populate itself with information contained in the active datastream received from the first computer system.

9. The computer system of claim 8 further comprising:

means for processing the instance of the datastream class by invoking at least one object method on the instance.

10. The computer system of claim 8 further comprising:

means for sending the instance of the datastream class by invoking at least one object method on the instance.

11. A method for communicating between a first computer system and a second computer system, the method comprising the steps of:

the first computer system constructing an active datastream, the active datastream including a datastream identifier that identifies executable code for processing the active datastream;

the first computer system sending the active datastream to the second computer system;

the second computer system creating an instance of a datastream class that corresponds to the datastream identifier; and

the instance of the datastream class populating itself with information contained in the active datastream received from the first computer system by invoking at least one object method on the instance.

12. The method of claim 11 further comprising the step of executing the executable code on the datastream instance to process the active datastream.

13. The method of claim 11 wherein the step of the instance of the datastream class populating itself with the information contained in the active datastream includes the step of executing a receive method on the instance of the datastream class.

14. The method of claim 11 wherein the step of the first computer system sending the active datastream to the second computer system includes the step of invoking at least one object method on the active datastream.

15. A method for communicating between a first computer system and a second computer system, the method comprising the steps of:

the first computer system constructing an active datastream object, the active datastream object including a datastream identifier that identifies a corresponding datastream class that includes executable code corresponding to a plurality of object methods for processing the active datastream object;

the first computer system sending the active datastream to the second computer system by invoking a send method on the active datastream object;

the second computer system reading the datastream identifier from the active datastream object received from the first computer system;

the second computer system creating a new instance of the datastream class that corresponds to the datastream identifier;

the new instance of the datastream class populating itself with information contained in the active datastream received from the first computer system by invoking a receive method on the new instance; and

the second computer system performing a request represented by the active datastream by invoking at least one object method on the new instance.

16. A program product comprising:

a datastream factory that creates an instance of a datastream class corresponding to an identifier in a received datastream;

a datastream receive mechanism defined in the datastream class that causes the instance of the datastream class to populate itself with information contained in the received datastream by invoking at least one object method on the instance; and

signal bearing media bearing the datastream factory and the datastream receive mechanism.

17. The program product of claim 16 wherein the signal bearing media comprises recordable media.

18. The program product of claim 16 wherein the signal bearing media comprises transmission media.

19. The program product of claim 16 further comprising a datastream processing mechanism on the signal bearing media for processing the instance of the datastream by invoking at least one object method on the instance.

20. The program product of claim 16 further comprising a datastream send mechanism on the signal bearing media for sending the instance of the datastream by invoking at least one object method on the instance.

OIPE JC37
NOV 1 0 2003
PATENT & TRADEMARK OFFICE

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

AF 2126 ~~2128~~

Appeal Brief Transmittal

In re application of: Nock, *et al.*

Serial No.: 09/259,361

Filed on: 02/26/99

For: **APPARATUS AND METHOD FOR COMMUNICATING BETWEEN COMPUTER SYSTEMS USING ACTIVE DATASTREAMS**

RECEIVED
NOV 1 4 2003
Technology Center 2100

Mail Stop APPEAL BRIEF - PATENT
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

Transmitted herewith for filing is an **Appeal Brief** in triplicate for the above-identified Application.

[X] Please deduct $330.00 from Deposit Account No. 09-0465 for IBM Corporation to cover the fee under 37 C.F.R. §1.17(f) for the filing of the enclosed appeal brief. A duplicate copy of this sheet is enclosed.

[X] The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 09-0465 for IBM Corporation. A duplicate copy of this sheet is enclosed.

- [X] Any additional filing fees required under 37 C.F.R. §1.16.
- [X] Any patent application processing fees under 37 C.F.R. §1.17.

MARTIN & ASSOCIATES, L.L.C.
P.O. Box 548
Carthage, MO 64836-0548
(417) 358-4700
FAX (417) 358-5757

Respectfully submitted,

By [illegible]
Derek P. Martin
Reg. No. 36,595

I HEREBY CERTIFY THAT THE CORRESPONDENCE TO WHICH THIS STATEMENT IS AFFIXED IS BEING DEPOSITED WITH THE UNITED STATES POSTAL SERVICE, POSTAGE PAID, AS FIRST CLASS MAIL IN AN ENVELOPE ADDRESSED TO: MAIL STOP APPEAL BRIEF - PATENT, COMMISSIONER FOR PATENTS, P.O. BOX 1450, ALEXANDRIA, VA 22313-1450.

Date: November 5, 2003 By: [illegible]